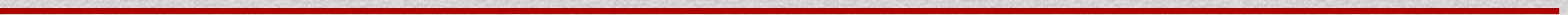
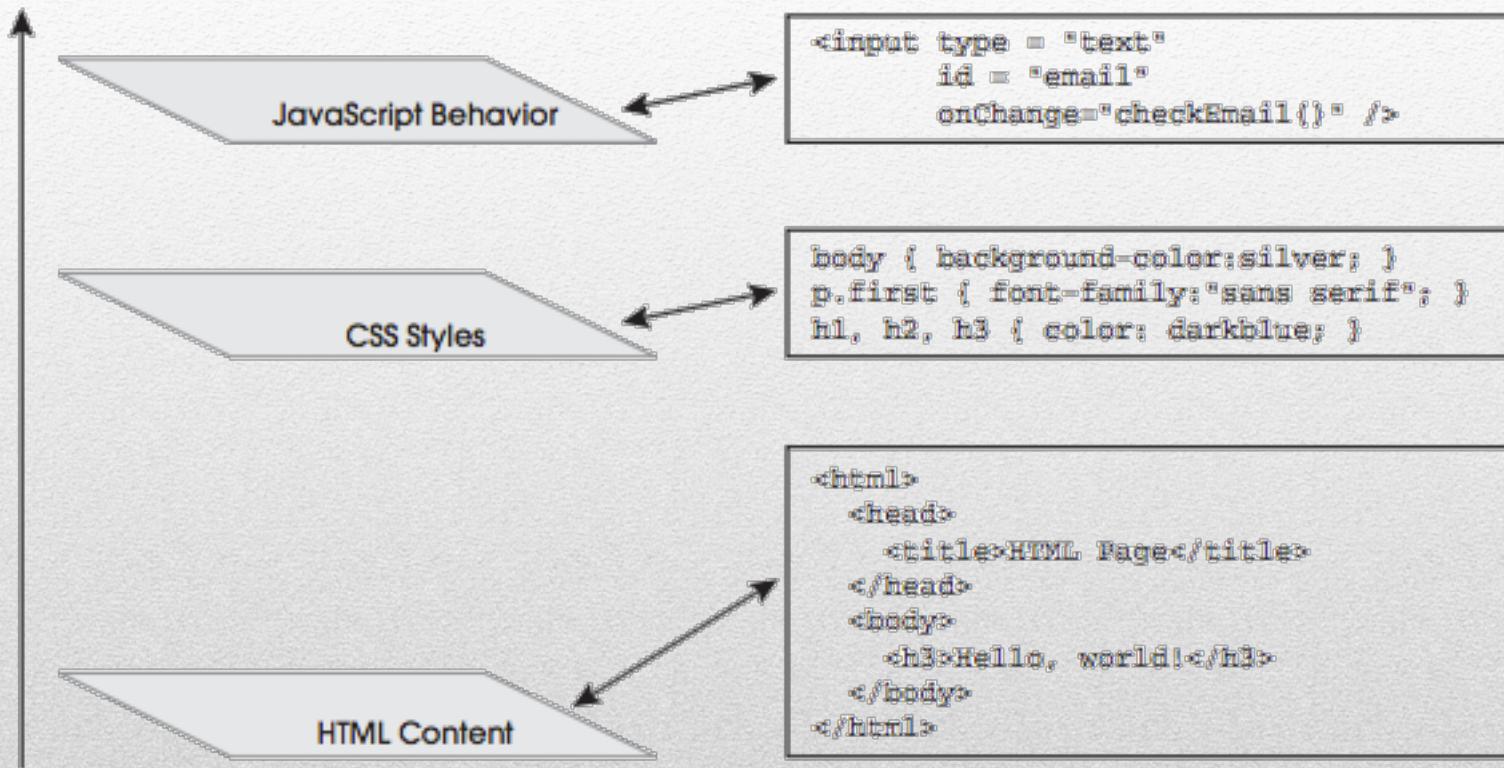


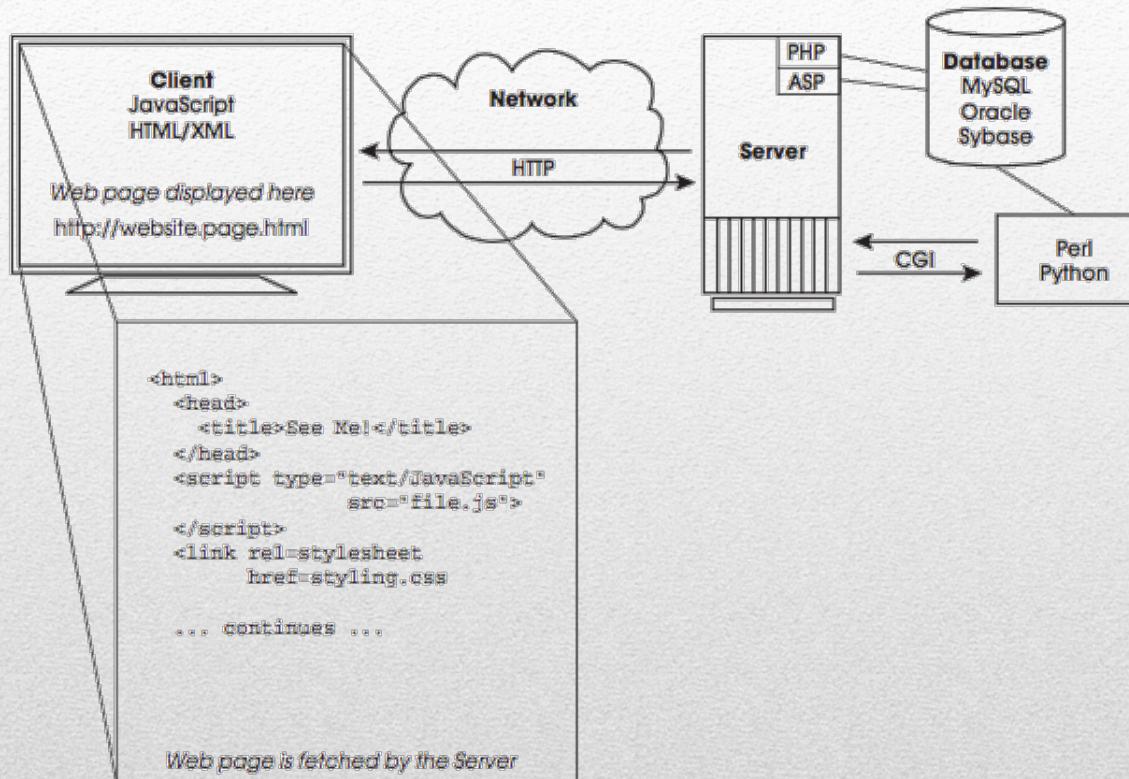
Document Object Model DOM



Página Web: *Layers*

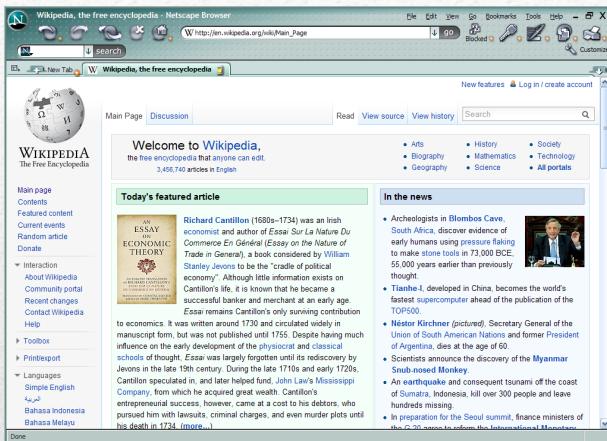


Página Web: Arquitetura



1. Utilizador faz um pedido de uma página através do URL
2. Servidor Web responde retornando o ficheiro HTML
3. Browser faz a renderização da página

Página Web: Document Object Model (DOM)



renderização



1. *Parsing* da página HTML (incluindo CSS) para construir a árvore DOM
2. Construção da árvore utilizada para renderização (render tree)
3. Construção do *layout* (localização dos elementos) definido na árvore de renderização
4. Atribuir as características visuais (a todos os nós da árvore DOM) definidos na árvore de renderização

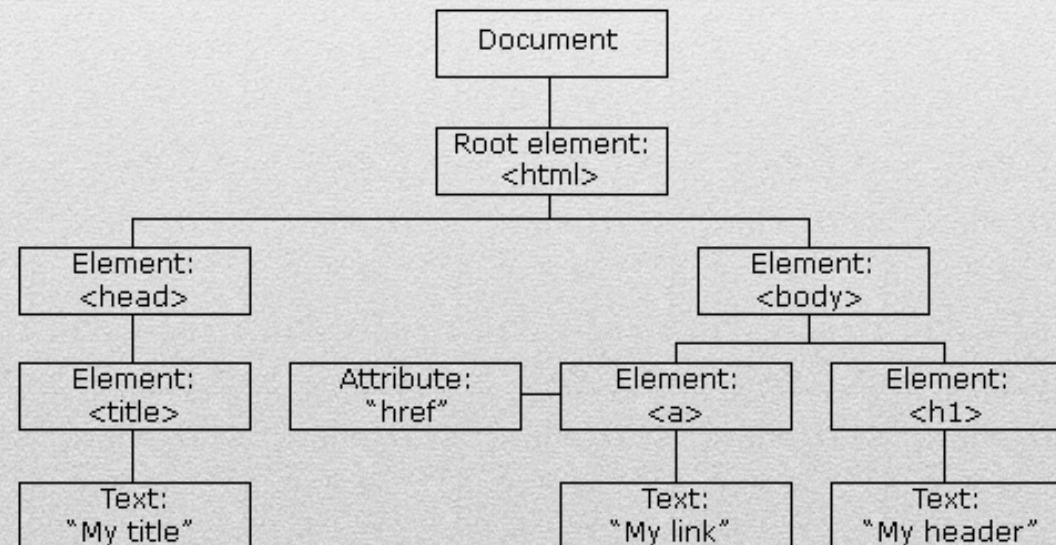
Página Web: HTML DOM

DOM tree (árvore de elementos)

```
<html>
  <head>
    <title>My title</title>
  </head>
  <body>
    <h1>My header</h1>
    <a href="url">My link</a>
  </body>
</html>
```

Página Web é um documento que pode ser visualizado no Browser ou como um ficheiro HTML

→ DOM providênciia uma outra representação estruturada em árvore do documento que pode ser acedido e alterado programmaticamente



Página Web: Acesso ao DOM

- “**document**” e “**window- **window**: objeto que representa a janela que contém o documento DOM carregado. Implementa a interface **Window**
- **document**: objeto que representa a página carregada no *browser* e serve que ponto de entrada para o conteúdo da página. Implementa a interface **Document****



API DOM

■ O que é o DOM?

- Plataforma standard do W3C e uma linguagem de interface neutra que permite que programas e *scripts* possam dinamicamente aceder e atualizar o conteúdo, a estrutura e o estilo de um documento
- Composto por 3 partes
 - Core DOM: modelo standard para todos os tipos de documento
 - XML DOM: modelo standard para documentos XML
 - HTML DOM: modelo standard para documentos HTML
- Recomendações
 - DOM Level 1
 - DOM Level 2
 - DOM Level 3
 - DOM Level 4

HTML DOM

- Modelo de objetos e uma interface de programação para HTML
 - Define os elementos HTML como objetos
 - Define propriedades de todos os elementos HTML
 - Define os métodos para aceder a todos os elementos HTML
 - Define eventos para todos os elementos HTML
- Standard para aceder, mudar, adicionar e eliminar elementos HTML



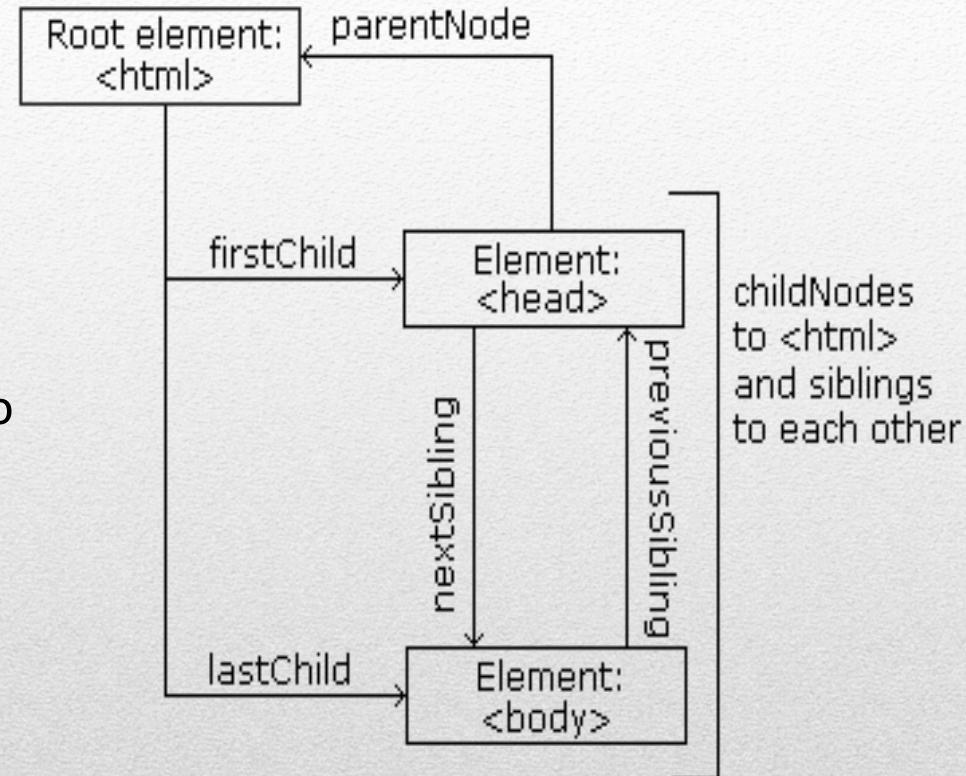
HTML DOM: Nós

- **Nó**
 - Documento é um nó documento
 - Elementos HTML são nós elemento da árvore
 - Atributos HTML são nós atributo
 - Texto dentro de elementos HTML são nós de texto
 - Comentários são nós de comentários
- HTML é carregado no *browser* e torna-se ***document object***
 - ***Document object*** é o nó *root* do HTML



HTML DOM: Elementos da Árvore DOM

- Primeiro nó designa-se por ***root***
- Todos os nós têm um nó ***parent***
- Cada nó pode ter um número qualquer de ***filhos***
- ***Siblings*** são nós com o mesmo ***parent***



HTML DOM: Propriedades de um Nó (objeto)

■ Propriedades mais comuns

- **x.innerHTML** – texto do elemento x
- **x.nodeName** – nome do elemento x
- **x.nodeValue** – valor do elemento x
- **x.parentNode** – nó *parent* do elemento x
- **x.childNodes** – nós filhos do elemento x
- **x.attributes** – nós atributo do elemento x
- **x.style.property** – aceder/alterar estilo do elemento x

Nota: no DOM 4, o objeto “Attr” deixa de herdar as características do objeto “Node”.

Devem ser utilizadas as propriedades e métodos do objeto “Attr”. Por exemplo, “value” em vez de `nodeValue`

HTML DOM: Interface “document” (I)

- **Encontrar elementos HTML no DOM (árvore)**
 - **document.getElementById()**
 - Encontrar elemento pelo seu Id
 - **document.getElementByTagName()**
 - Encontrar elemento pela sua *Tag*
 - **document.getElementByClassName()**
 - Encontrar elemento pelo nome da classe
 - **document.querySelectorAll()**
 - Encontrar elementos pelo seletor CSS

A interface “document” representa a página carregada no *browser* e serve como uma forma de aceder ao conteúdo da página

HTML DOM: Interface “document” (II)

- **Adicionar e Apagar elementos HTML no DOM (árvore)**
 - **document.createElement()** – criar elemento
 - **document.removeChild()** – remover elemento
 - **document.appendChild()** – adicionar elemento
 - **document.replaceChild()** – substituir elemento



HTML DOM: Eventos

■ Elementos HTML

...

```
<body>
    <button onclick="displayDate" id="myBtn">Try it</button>
</body>
```

...

■ HTML DOM

...

```
<button id="myBtn">Try it</button>
<p id="demo"></p>
```

```
<script>
```

```
    document.getElementById("myBtn").onclick = displayDate;
```

```
    function displayDate() {
```

```
        document.getElementById("demo").innerHTML = Date();
```

```
    }
```

```
</script>
```

...

HTML DOM: Eventos – addEventListener

Sintaxe

```
element.addEventListener(eventname, function, useCapture);
```

Função que queremos chamar quando
o evento ocorrer

Tipo de evento, por exemplo: "click" ou "mousedown"
NOTA: Não se utiliza o prefixo "on" ("click" em vez de
"onclick")

useCapture= FALSE → "Event bubbling" (default)
useCapture= TRUE → "Event capturing"

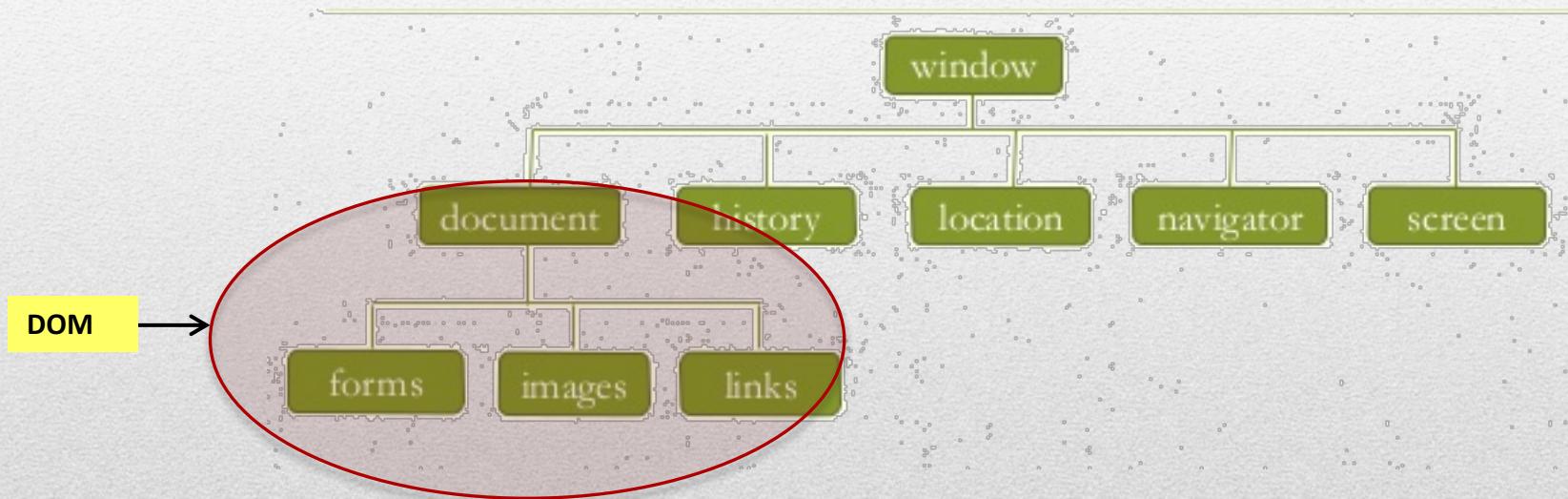
Exemplo

```
document.getElementById("myBtn").addEventListener("click", displayDate);
```

```
document.getElementById("myBtn").removeEventListener("click", displayDate);
```

Browser Object Model (BOM)

- Permite que o JavaScript “comunique” com o browser (não é oficial)



- Objetos, funções e variáveis Globais em JavaScript tornam-se membros do objeto **Window**

JavaScript: Documento HTML

- *Inline*

```
<nav>
  <ul>
    <li><a href="/about" onclick="alert('this is the thing');">
      About</a></li>
    <li><a href="/articles">Articles</a></li>
    <li><a href="/staff">Staff</a></li>
    <li><a href="/contact">Contact</a></li>
  </ul>
</nav>
```

JavaScript Inline

↓

JavaScript: Documento HTML

- *Embedded*

```
<html>
```

```
<head>
```

```
    <script>
```

```
        function product(a,b)
        {
        }
    
```

```
    </script>
```

```
</head>
```

```
<body>
```

```
    <script>
```

```
        document.write(product(4,3));
    
```

```
    </script>
```

```
    <p>The script calls a function with two parameters (4 and
3).</p>
```

```
</body>
```

```
</html>
```

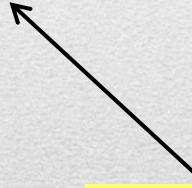
JavaScript
Incorporado no
HTML

JavaScript: Documento HTML

- Ficheiro Externo

```
...
<head>
    <meta charset="UTF-8">
    <title>Questionário</title>
    <link rel="stylesheet" href="css/questionario.css"
        type="text/css"/>
    <script src="js/questionario.js"></script>
</head>
<body>
<header>
    <h1>Questionário</h1>
    <h2>Google Images</h2>
</header>
<section id="intro">
    ...

```



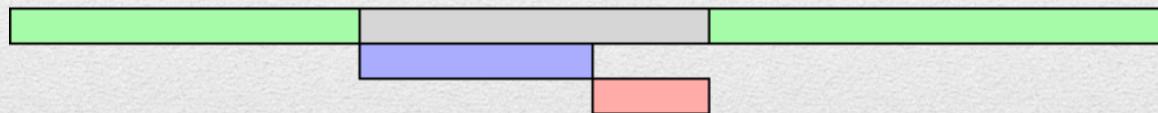
JavaScript num ficheiro separado
torna mais perceptível o html

JavaScript: <script>

■ Atributos

```
<script src="js/questionario.js"></script>
```

- “async = false” e “defer = false” **(sem atributos)**



- █ HTML parsing
- █ HTML parsing paused
- █ Script download
- █ Script execution

1 - *Script* é executado imediatamente após ser feito *parsing/load* do código

2 - É interrompido o *parsing/load* do ficheiro HTML

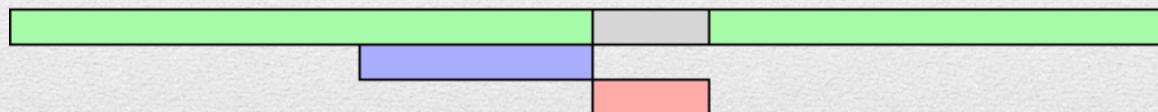
3 - *Script* é executado antes que o browser continue a fazer o *parsing* da página

JavaScript: <script>

■ Atributos

```
<script src="js/questionario.js" async></script>
```

- “async = true”



- HTML parsing
- HTML parsing paused
- Script download
- Script execution

1 - Script é carregado de forma assíncrona, enquanto o browser faz o *parsing* da página HTML.

2 - A seguir é feita uma pausa no *parsing* da página para executar o *script*

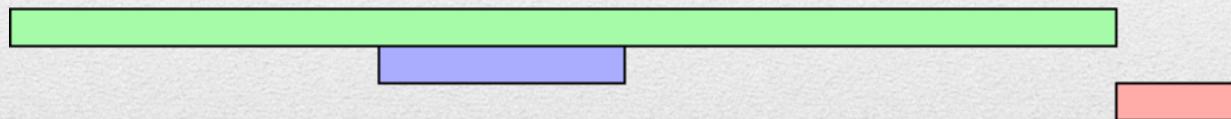
3 - Não garante a ordem de execução. Script incluído no fim pode ser executado primeiro que um *script* incluído no início

JavaScript: <script>

■ Atributos

```
<script src="js/questionario.js" defer></script>
```

- “defer = true”



- HTML parsing
- HTML parsing paused
- Script download
- Script execution

1 - *Script* é carregado enquanto o browser faz o *parsing* da página.

2 - *Script* é executado após o *parsing* completo da página.

3 - Este atributo garante que o *script* é executado pela ordem que aparece no documento. *Script* incluído no fim é executado depois que um *script* incluído no início